ProRealTime SOFTWARE

Guide de Programmation

Fonctions de Base & Indicateurs (ProBuilder)

V 4.0.0 - 20140916

SOMMAIRE

Présentation de ProBuilder 1

Chapitre I : Les notions fondamentales 2

- Utiliser ProBuilder 2
  - Création d'indicateur 2
  - Raccourcis clavier de l'éditeur de code 5
- Spécificités de programmation du langage ProBuilder 6
- Les constantes financières ProBuilder 7
  - Les constantes de prix et de volume adaptées à l'unité de temps du graphique 7
  - Les constantes journalières de prix 8
  - Les constantes temporelles 8
  - Les constantes dérivées des prix 12
  - La constante indéfinie 12
- Utilisation des indicateurs préexistants 12
- Optimisation des variables 13

Chapitre II : Fonctions et instructions ProBuilder 15

- Structures de contrôle 15
  - Instruction conditionnelle IF 15
    - Une condition, un résultat (IF THEN ENDIF) 15
    - Une condition, deux résultats (IF THEN ELSE ENDIF) 15
    - Conditions imbriquées 15
    - Conditions Multiples (IF THEN ELSIF ELSE ENDIF) 16
  - Boucle itérative FOR 17
    - Avancement croissant (FOR, TO, DO, NEXT) 17
    - Avancement décroissant (FOR, DOWNTO, DO, NEXT) 18
  - Boucle conditionnelle WHILE 19
  - BREAK 20
    - Avec WHILE 20
    - Avec FOR 20
  - CONTINUE 21
    - Avec WHILE 21
    - Avec FOR 21
  - ONCE 22
- Fonctions Mathématiques 23
  - Fonctions usuelles unaires et binaires 23
  - Opérateurs mathématiques usuels 23
  - Fonctions de comparaisons graphiques 23
  - Fonctions de sommation 24
  - Fonctions Statistiques 24
- Opérateurs logiques 24

- Instructions ProBuilder....24
  - RETURN....25
  - REM ou //....25
  - CustomClose....25
  - CALL....26
  - AS....26
  - COLOURED....26

Chapitre III : Applications pratiques____28

- Créer un indicateur binaire ou ternaire : pourquoi et comment ?....28
- Créer des indicateurs STOP : suivez vos positions en temps réel....29
  - STOP prise de bénéfices statique....30
  - STOP loss statique....30
  - STOP d'inactivité....31
  - STOP suiveur ou trailing stop....32

Chapitre IV : exercices____33

- Figures de chandeliers....33
- Indicateurs....34

Glossaire____36

Avertissement : ProRealTime n'exerce pas le service de Conseil en Investissement Financier. Ce document n'est en aucun cas une offre de conseil en investissement ni une incitation quelconque à acheter ou vendre des instruments financiers. Les exemples présentés dans ce manuel sont à but pédagogique. Pour votre propre trading, vous êtes entièrement libre dans le choix de vos critères. Les performances passées ne présagent pas de l'avenir. Tout système de trading peut vous exposer à un risque de perte supérieur à votre investissement initial.

Présentation de ProBuilder

ProBuilder est le langage de programmation de ProRealTime. Ce dernier sert à concevoir des indicateurs techniques personnalisés, des stratégies de trading (ProBackTest) ou des scans personnalisées (ProScreener). ProBackTest et ProScreener font l'objet de manuels individuels à cause de certaines spécificités de programmation.

Ce langage est de type BASIC, très simple d'utilisation et exhaustif dans les possibilités offertes.

Vous allez pouvoir construire vos propres programmes qui utilisent les cotations de n'importe quel instrument inclus dans l'offre ProRealTime, à partir des éléments de base :

- le cours d'ouverture de chaque barre : Open
- le cours de clôture, de chaque barre : Close
- le plus haut de chaque barre : High
- le plus bas de chaque barre : Low
- le nombre de titres échangés : Volume.

Les barres, ou chandeliers, sont les représentations graphiques standards des cotations reçues en temps réel. ProRealTime vous offre bien entendu la possibilité de personnaliser le type de style graphique, vous proposant, parmi d'autres, vues telles que Renko, Kagi, Heikin-Ashi.

ProBuilder évalue les données de chaque barre de prix depuis la plus ancienne jusqu'à la plus récente, et exécute la formule développée dans le langage afin de déterminer la valeur des indicateurs sur la barre en question.

Les indicateurs développés sous ProBuilder peuvent être affichés sur le graphique du prix ou bien dans un graphique individuel, selon le type d'échelle utilisée.

Dans ce document, vous assimilerez au fur et à mesure les commandes permettant de programmer dans ce langage grâce une vision théorique claire et des exemples concrets les illustrant.

A la fin de ce manuel, vous y trouverez un Index qui vous donnera une vue de l'ensemble des commandes de ProBuilder, des indicateurs déjà codés et d'autres fonctions complétant ce que vous aurez appris durant votre lecture.

Les utilisateurs plus habitués à la programmation, pourront passer directement à la lecture du chapitre II ou bien consulter l'index afin de retrouver rapidement l'explication relative à la fonction recherchée.

Pour ceux moins habitués à programmer, nous conseillons le visionnage de la vidéo intitulée "**Créer un indicateur dans ProBuilder**" et de lire l'intégralité du manuel. Très directif et fortement orienté vers la pratique, nous ne doutons pas que vous serez en mesure de maîtriser ce langage en peu de temps.

En vous souhaitant nos meilleurs voeux de réussite, Bonne lecture !

Chapitre I : Les notions fondamentales

Utiliser ProBuilder

Création d'indicateur

La zone de programmation d'un indicateur est disponible à partir du bouton "Indicateur/Backtest" qui se trouve en haut à droite dans chaque graphique de votre plateforme ProRealTime.

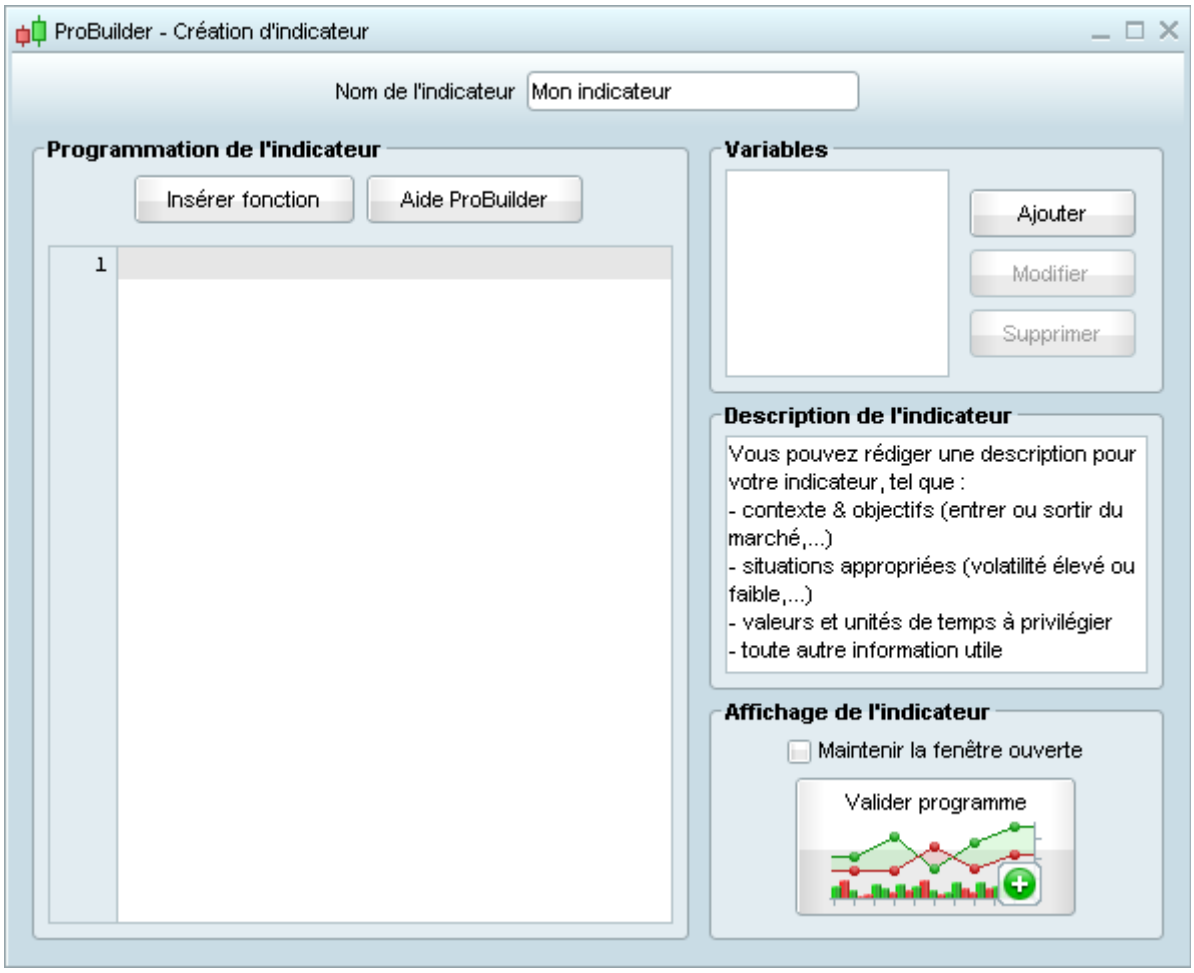
Vous accéderez ensuite à la fenêtre de gestion des indicateurs. Vous pourrez :

- Afficher un indicateur prédéfini.
- Créer un indicateur personnalisé, qui pourra ensuite être appliqué à n'importe quelle valeur.

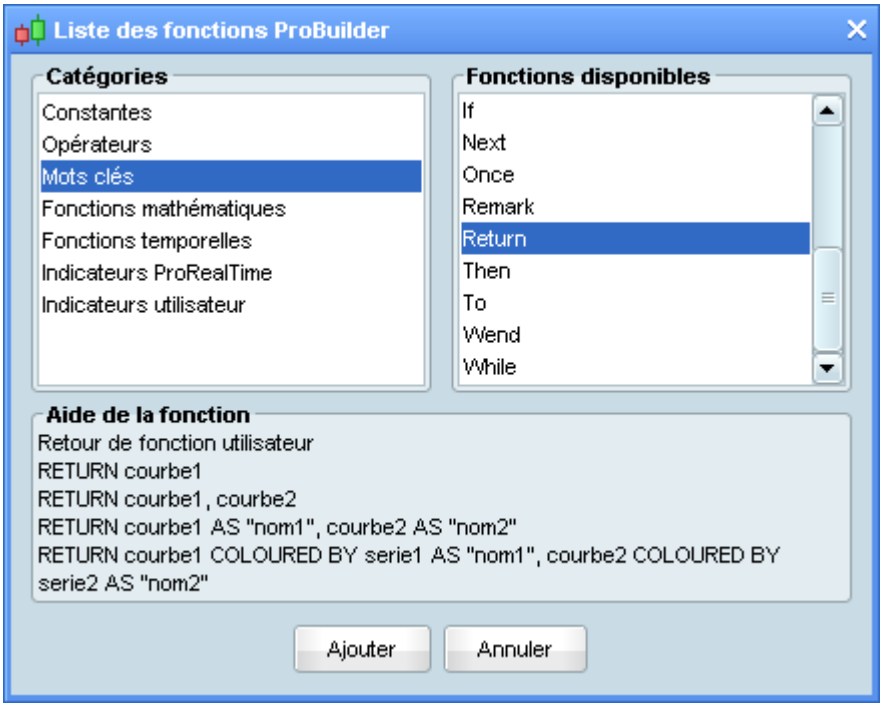
Dans le second cas cliquez sur "Créer indicateur", pour accéder à la fenêtre de programmation.

Vous avez alors la possibilité :

- de programmer directement un indicateur dans la zone de texte réservée au code.
- d'utiliser la fonction d'aide "Insérer Fonction", qui permet de trouver dans une nouvelle fenêtre une bibliothèque des fonctions disponibles, séparées en sept catégories, afin de vous assister lors de la programmation.



Prenons comme exemple le premier élément caractéristique des indicateurs ProBuilder, c'est-à-dire la fonction "**RETURN**", (disponible dans la section "Mots clés" - voir image ci-dessous).

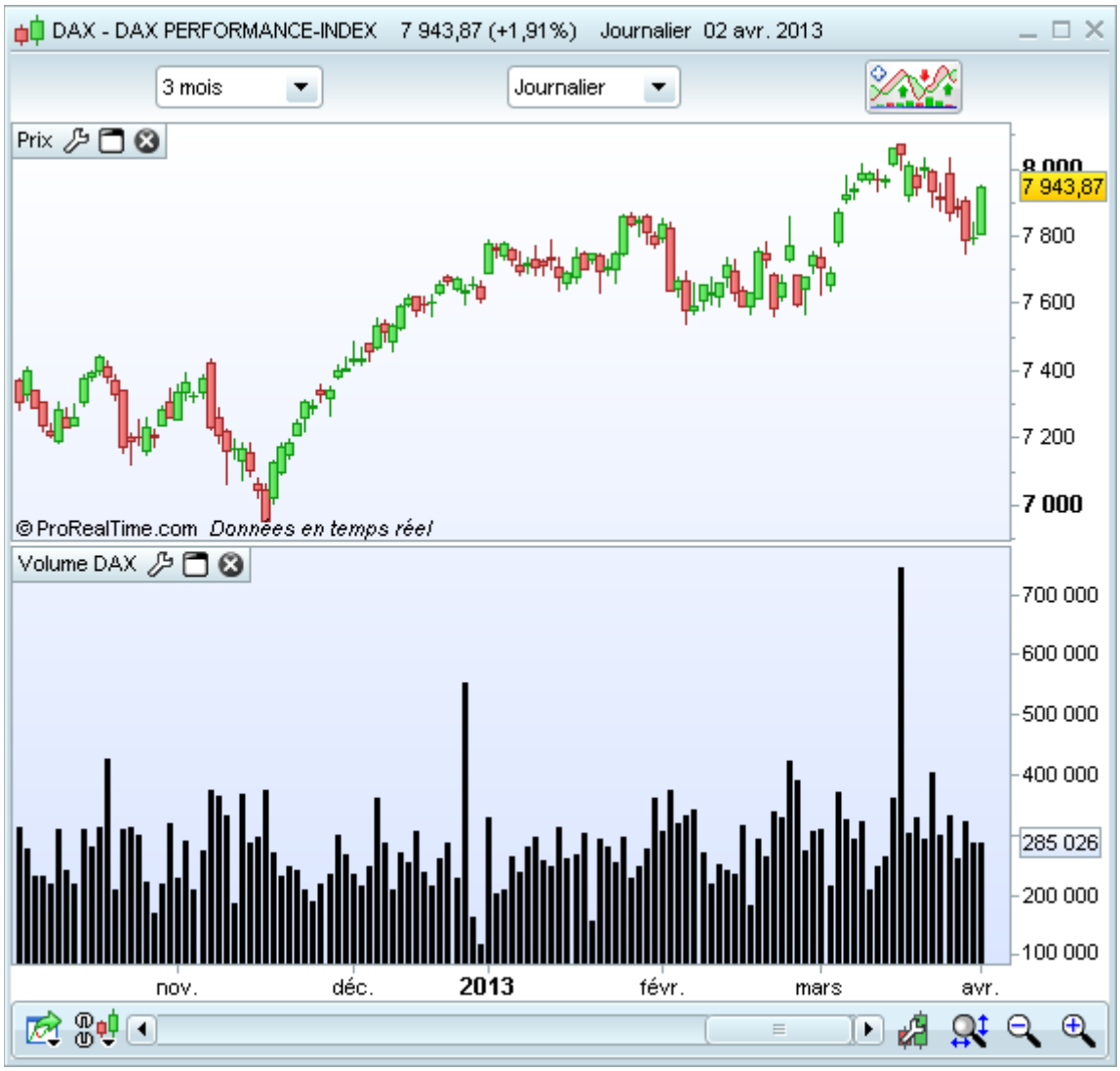


Sélectionnez donc le mot "**RETURN**" et cliquez sur "Ajouter" : la commande s'ajoutera bien à la zone de programmation.

RETURN vous permet d'afficher le résultat

Supposons que l'on veuille créer un indicateur affichant le Volume. Si vous avez déjà inséré le mot **RETURN**, il suffit alors d'aller une nouvelle fois sur "Insérer fonction", de cliquer sur "Constantes" dans la section "Catégories", puis sur le côté droit, section "Fonctions disponibles", cliquer sur "**Volume**". Enfin, cliquez sur "Ajouter".

Avant de cliquer sur le bouton "Valider programme", précisez en haut de la fenêtre le nom de votre indicateur : ici, nous l'avons appelé "Volume DAX". Enfin, cliquez sur "Valider programme" et vous verrez s'afficher le graphique de votre indicateur.



Raccourcis clavier de l'éditeur de code

Dans ProRealTime version 10, la fenêtre de création de systèmes de trading possède plusieurs fonctionnalités pratiques utilisables via des raccourcis clavier :

- Tout sélectionner (Ctrl+A) : Sélectionne tout le texte présent dans l'éditeur de code
- Copier (Ctrl + C) : Copie le texte sélectionné
- Coller (Ctrl + V) : Colle le texte copié
- Annuler (Ctrl + Z) : Annule la dernière action faite dans l'éditeur de code
- Refaire (Ctrl + Y) : Refait la dernière action faite dans l'éditeur de code
- Rechercher / Remplacer (Ctrl + F) : Cherche un texte dans l'éditeur de code / Remplace un texte dans l'éditeur de code (cette fonctionnalité est sensible à la casse)
- Commenter / Dé-commenter (Ctrl + R) : Commente le code sélectionné / Dé-commente le code sélectionné (le code commenté sera précédé de "//" ou de "REM" et coloré en gris. Il ne sera pas pris en compte lors de l'exécution du code.

Pour les utilisateurs de MAC, les mêmes raccourcis claviers peuvent être utilisés. Il suffit dans ce cas de remplacer la touche "Ctrl" par la touche " Pomme".

La plupart de ces raccourcis peuvent aussi être utilisé via un clic droit dans l'éditeur de code de la fenêtre de création du système de trading.

Spécificités de programmation du langage ProBuilder

Les spécificités

Le langage ProBuilder vous permet de manipuler de nombreuses commandes classiques ainsi que des outils plus élaborés spécifiques à l'analyse technique, qui vous donneront la possibilité de programmer des indicateurs du plus simple au plus sophistiqué.

Les principes clés à connaître sur le langage ProBuilder sont :

- Il n'est **pas nécessaire de déclarer les variables**.
- Il n'est **pas nécessaire de typer les variables**.
- Il n'y a **pas de différence entre majuscule et minuscule** (il existe par contre une particularité que nous allons aborder juste après).
- On utilise le même symbole **pour l'affectation et l'égalité mathématique**.

Qu'est-ce que cela signifie ?

- Déclarer une variable X, c'est indiquer son existence. Dans ProBuilder, vous pouvez directement utiliser X sans avoir défini antérieurement son existence.Prenons un exemple en écrivant :

  Avec déclaration : Soit la variable X, On attribue à X la valeur 5

  Sans déclaration : On attribue à X la valeur 5 (donc implicitement, X existe et vaut 5)

  En ProBuilder il suffit d'écrire : X=5

- Typer une variable, c'est à dire définir la nature de la variable : est-elle un entier naturel (ex : 3 ; 8 ; 21 ; 643 ; …), un entier relatif (ex : 3 ; 632 ; -37 ; …), un décimal (ex : 1.76453534535…), un booléen (VRAI, FAUX),… ?

- Dans ProBuilder, vous pouvez écrire vos commandes aussi bien avec des majuscules qu'avec des minuscules. Par exemple, l'ensemble de commandes IF / THEN / ELSE / ENDIF pourra indifféremment être écrite iF / tHeN / ELse / endIf.

  Exception : Quand vous décidez d'utiliser une variable, il faudra que vous respectiez la casse du nom défini. Si vous avez écrit : "vARiaBLe" et vous souhaitez la réutiliser dans le programme, seul le respect de la casse permettra de se référer à la même variable "vARiaBLe"

- Affecter une valeur à une variable, c'est lui attribuer une valeur. Pour mieux comprendre le principe d'affectation, il faut que vous considériez une variable comme une boîte vide qui attend qu'on lui mette quelque chose à l'intérieur. Le schéma ci-dessous vous illustre ce principe avec la valeur Volume affectée à la variable X :

X ⟵ Volume

Voyez bien qu'on lit de droite à gauche la phrase : Volume est affecté à X.

Maintenant, pour l'écrire en code ProBuilder, on va simplement remplacer la flèche par un signe =

X = Volume

Le même symbole = est utilisé :

- pour l'affectation d'une variable (comme dans l'exemple précédent).
- comme opérateur mathématique binaire (1+ 1= 2 est équivalent à 2 = 1 + 1).

Les constantes financières ProBuilder

Avant de commencer à coder vos indicateurs personnels, il est nécessaire de passer en revue les éléments à partir desquels vous pourrez constituer votre code, tels que les prix d'ouverture et de clôture, le volume, etc...

Ce sont les "fondamentaux" de l'analyse technique, et l'essentiel à connaître pour coder des indicateurs.

Vous pourrez ainsi les combiner afin de faire ressortir certains aspects de l'information fournie par les marchés financiers. On peut les regrouper en 5 catégories :

Les constantes de prix et de volume adaptées à l'unité de temps du graphique

Ce sont les constantes "classiques" les plus utilisées. Elles reportent par défaut les valeurs de la barre en cours (quelque soit l'unité de temps du graphique) et se présentent de la façon suivante :

- **Open :** le cours d'ouverture de la barre courante.
- **High :** le cours le plus haut de la barre courante.
- **Low :** le cours le plus bas de la barre courante.
- **Close :** le cours de clôture de la barre courante.
- **Volume :** le nombre de titres ou de lots échangés sur la barre courante.

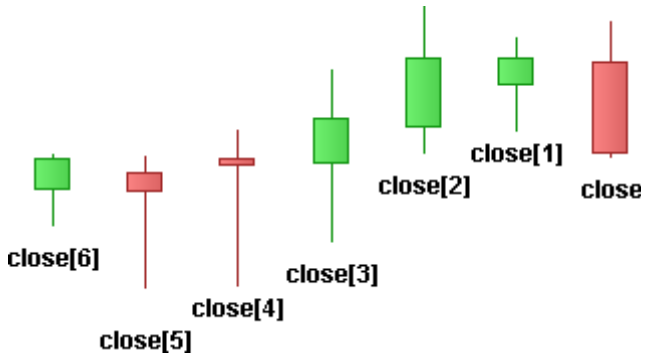
Exemple : Range de la barre courante

```
a = High
b = Low
MyRange = a - b
RETURN MyRange
```

Pour faire appel aux valeurs des barres précédentes, il suffit de rajouter entre crochets le chiffre relatif à la barre à considérer (le nombre de barres à partir de la barre courante).

Prenons par exemple la constante prix de clôture. L'appel du cours se fait de la façon suivante :

Valeur de la fermeture de la barre courante :	*Close*
Valeur de la fermeture de la barre qui précède la courante :	*Close[1]*
Valeur de la fermeture de la n-ième barre qui précède la courante :	*Close [n]*



Cette règle vaut pour n'importe quelle constante. Par exemple, le prix d'ouverture de la 2ème barre précédent la barre courante, sera appelé par : Open[2].

La valeur qui sera reportée dépendra de la période affichée sur le graphique.

Les constantes journalières de prix

Contrairement aux constantes adaptées à l'unité de temps du graphique, les constantes journalières de prix se réfèrent aux valeurs de la journée, indépendamment de la période affichée sur le graphique.

Une autre différence à constater avec les constantes adaptées à l'unité de temps, est que les constantes journalières utilisent des parenthèses pour obtenir leurs valeurs sur des barres antérieures.

- **DOpen(n) :** prix d'ouverture de la nième journée antérieure à celle de la barre courante.
- **DHigh(n) :** prix le plus haut de la nième journée antérieure à celle de la barre courante.
- **DLow(n) :** prix le plus bas de la nième journée antérieure à celle de la barre courante.
- **DClose(n) :** prix de clôture de la nième journée antérieure à celle de la barre courante.

Remarque : si "n" est égal à 0, on récupère alors la valeur du jour comme montrée dans l'exemple ci-dessous.

Exemple : Range journalier

```
a = DHigh(0)
b = DLow(0)
MyRange = a - b
RETURN MyRange
```

Pour les constantes adaptées à l'unité de temps on utilise de crochet, pour les constantes journalières on utilise des parenthèses.

```
Close[3]
DClose(3)
```

Les constantes temporelles

Le temps est une composante parfois négligée de l'analyse technique. Pourtant les traders connaissent bien l'importance de certains moments de la journée, ou de certaines dates de l'année. Il est donc possible de limiter l'analyse de son indicateur à des moments spécifiques en utilisant les constantes suivantes :

- **Date :** Date codée sous la forme YYYYMMJJ indiquant la date de clôture de chaque barre.

Les constantes temporelles sont considérées par ProBuilder comme des entiers. La constante Date, par exemple, doit être présentée comme un unique nombre composé de 8 chiffres.

Ecrivons alors le programme :

```
RETURN Date
```

Supposons que nous soyons aujourd'hui le 4 juillet 2008. L'indicateur issu du programme ci-dessus va relier toutes les dates de clôture des barres et renvoyer le résultat 20080704.

Pour lire une date, il suffit donc de lire de la manière suivante :

20080704 = 2008 années 07 mois et 04 jours.

Attention, dans l'écriture d'une date au format YYYYMMJJ, MM ne doit pas dépasser la valeur 12 et JJ ne doit pas dépasser la valeur 31.

- **Time :** HeureMinuteSeconde codée sous la forme HHMMSS indiquant l'heure de clôture de chaque barre.

Faisons par exemple :

```
RETURN Time
```

On obtient une courbe liant toutes les heures de clôture de chaque barre :

Pour lire une heure, il suffit donc de lire de la manière suivante :

160000 = 16 heures 00 minutes et 00 secondes.

Attention, dans l'écriture d'une heure au format HHMMSS, HH ne doit pas dépasser la valeur 23, MM ne doit pas dépasser la valeur 59 et SS ne doit pas dépasser la valeur 59.

Il est possible de combiner dans un même indicateur **Time** et **Date** afin de restreindre le résultat à un moment spécifique. Dans l'exemple qui suit, nous voulons limiter notre indicateur au premier octobre 2008 à 9h00 et 1 sec.

```
a = (Date = 20081001)
b = (Time = 090001)
RETURN (a AND b)
```

De la même façon fonctionnent les constantes suivantes :

- **Minute :** Minute de la clôture de chaque barre (entre 0 et 59).
- **Hour :** Heure de la clôture de chaque barre (entre 0 et 23).
- **Day :** Jour du mois de la clôture de chaque barre (entre 1 et 28 ou 29 ou 30 ou 31).
- **Month :** Mois de la clôture de chaque barre (entre 1 et 12).
- **Year :** Année de la clôture de chaque barre.
- **DayOfWeek :** Jour de la semaine à la clôture de chaque barre (ne traite pas les jours de Weekend-1=lundi, 2=mardi, 3=mercredi, 4=jeudi, 5=vendredi).

Exemple d'utilisation de ces constantes :

```
a = (Hour > 170000)
b = (Day = 30)
RETURN (a AND b)
```

- **CurrentHour :** Heure actuelle (celle du marché).
- **CurrentMinute :** Minute actuelle (celle du marché).
- **CurrentMonth :** Mois actuel (celle du marché).
- **CurrentSecond :** Seconde actuelle (celle du marché).
- **CurrentTime :** HeureMinuteSeconde actuelle (celle du marché).
- **CurrentYear :** Année actuelle (celle du marché).
- **CurrentDayOfWeek :** Jour de la semaine actuelle selon le fuseau horaire du marché.

La différence entre les constantes Current proposées ci-dessus et celles sans Current vues précédemment est justement le côté "Actuel".

L'image suivante met en évidence cette différence appliquée aux constantes **CurrentTime** et **Time**. Pour simplifier, les constantes en Current font abstraction de l'axe des temps et ne considère que la valeur affichée dans l'encadré blanc.

Time indique l'heure de clôture de chaque barre

CurrentTime indique l'heure du marché

Si vous souhaitez régler vos indicateurs par rapport à des compteurs (nombres de jours écoulés, nombre des barres etc…), les constantes Days, BarIndex et IntradayBarIndex sont à votre disposition.

- **Days :** Compteur de jours depuis 1900.

Cette constante est utile lorsqu'on souhaite connaitre le nombre de jours qui se sont écoulés, particulièrement lorsqu'on travaille en vues quantitatives, comme (x)Tick ou (x)Volumes.

L'exemple suivant permettra donc d'afficher le passage d'une journée à l'autre de cotations, lorsqu'on se trouve dans l'une des vues citées.

```
RETURN Days
```

(Attention à ne pas confondre les deux constantes "Day" et "Days").

- **BarIndex :** Compteur de barres depuis le début de l'historique affiché.

Le compteur part de la barre la plus à gauche de l'historique chargé et compte toutes les barres jusqu'à celle la plus à droite – barre en cours incluse. La première barre affichée (la plus à gauche) est considérée comme la barre 0 (contrairement au fonctionnement des autres constantes). **BarIndex** s'utilise dans la majorité des cas avec l'instruction **IF** présentée plus loin dans le manuel.

- **IntradayBarIndex :** Compteur de barres intraday.

Le compteur affiche le nombre de barres depuis le début de la journée et est réinitialisé à zéro tous les débuts de journée. La première barre du compteur est considérée comme la barre 0 (contrairement au fonctionnement des autres constantes).

Comparons donc les deux constantes en créant deux indicateurs séparés :

```
RETURN BarIndex
```

et

```
RETURN IntradayBarIndex
```

On remarque bien la remise à zéro du compteur de barres tous les débuts de journée pour IntradayBarIndex.

Les constantes dérivées des prix

- **Range :** différence entre High et Low.
- **TypicalPrice :** moyenne entre High, Low et Close.
- **WeightedClose :** moyenne pondérée de High (poids 1) Low (poids 1) et Close (poids 2).
- **MedianPrice :** la moyenne entre High et Low.
- **TotalPrice :** la moyenne entre Open, High, Low et Close.

Le **Range** représente la volatilité de la barre courante.

Le **WeightedClose** insiste sur l'importance du cours de clôture.

Les constantes **TypicalPrice** et **TotalPrice** reflètent mieux la psychologie du marché intra-barre courante car elles prennent en compte trois et quatre niveaux de cours atteints au cours de la journée.

Le **MedianPrice** représente le prix moyen des extrémums.

Range en % :

```
MyRange = Range
Calcul = (MyRange / MyRange[1] - 1) * 100
RETURN Calcul
```

La constante indéfinie

Undefined permet d'indiquer à l'indicateur de ne pas afficher un résultat pour certaines variables (par défaut toutes les variables non affectées sont à zéro).

- **Undefined :** donnée indéfinie (équivalent d'une case vide ou de NULL).

Vous pouvez retrouver un exemple d'application plus loin dans le manuel.

Utilisation des indicateurs préexistants

Nous avons jusqu'à présent observé les possibilités offertes par Probuilder en termes de constantes et leur comportement lors de l'accès aux barres du passé. Le même comportement s'applique au fonctionnement des indicateurs préexistants (et par la suite nous verrons que ceux que vous programmerez fonctionneront selon le même principe).

Les indicateurs ProBuilder se composent de trois éléments dont la syntaxe est :

NomDeFonction [calculé sur n barres] (sur tel prix ou tel indicateur)

Lorsque l'on utilise le bouton "Insérer Fonction" pour rechercher une fonction ProBuilder , des valeurs par défaut sont placées pour la période et pour l'argument prix ou indicateur

Average[20](Close)

On peut bien sûr les modifier selon nos préférences ; par exemple, on peut remplacer les 20 barres définies par défaut par 100 barres. De même, on peut modifier l'argument de prix ou indicateur "Close" par "**Dopen(6)**" :

Average[20](Dopen(6))

Regardons quelques exemples de comportement d'indicateurs préexistants :

Programme calculant la moyenne mobile exponentielle sur 20 bougies appliquée au prix de clôture :

```
RETURN ExponentialAverage[20](Close)
```

Calcul d'une moyenne mobile pondéré sur 20 barres appliquée au prix typique

```
mm = WeightedAverage[20](TypicalPrice)
RETURN mm
```

Calcul d'une moyenne mobile à lissage de Wilder sur 100 barres appliquée au Volume

```
mm = WilderAverage[100](Volume)
RETURN mm
```

Calcul du MACD (en histogramme) sur le prix de clôture. La ligne du MACD se construit comme la différence entre la moyenne mobile exponentielle sur 12 périodes moins celle sur 26 périodes. On effectue ensuite un lissage avec une moyenne mobile exponentielle à 9 périodes, appliquée à la différence pour obtenir la ligne de Signal. Le MACD en histogramme se constitue alors de la différence entre la ligne du MACD et la ligne de Signal.

```
REM Calcul de la ligne MACD
LigneMACD = ExponentialAverage[12](Close) - ExponentialAverage[26](Close)
REM Calcul de la ligne du signal MACD
LigneSignal = ExponentialAverage[9](LigneMACD)
REM Calcul de la différence entre la ligne du MACD et son Signal
MACDHistogramme = LigneMACD - LigneSignal
RETURN MACDHistogramme
```

Optimisation des variables

Lorsque l'on code un indicateur, on introduit un certain nombre de constantes. L'option d'optimisation des variables en haut à droite, vous permet d'attribuer une valeur par défaut à une constante indéfinie et d'agir ensuite sur la valeur de cette constante à partir de l'interface des paramètres de l'indicateur.

L'avantage réside dans la possibilité de pouvoir modifier les paramètres de l'indicateur sans modifier le code.

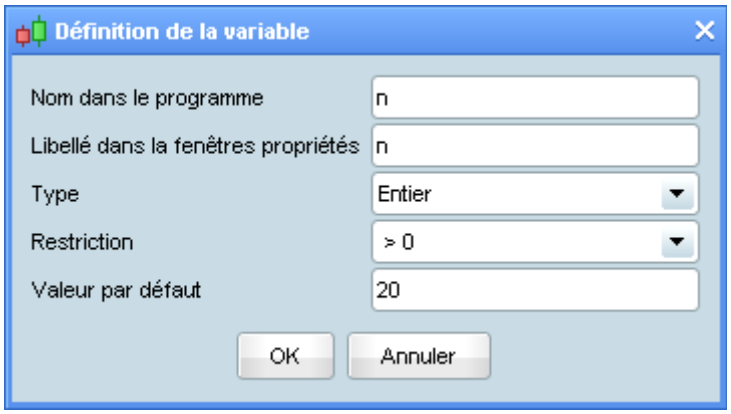
Calculons par exemple une moyenne mobile de période 20 :

```
RETURN Average[20](Close)
```

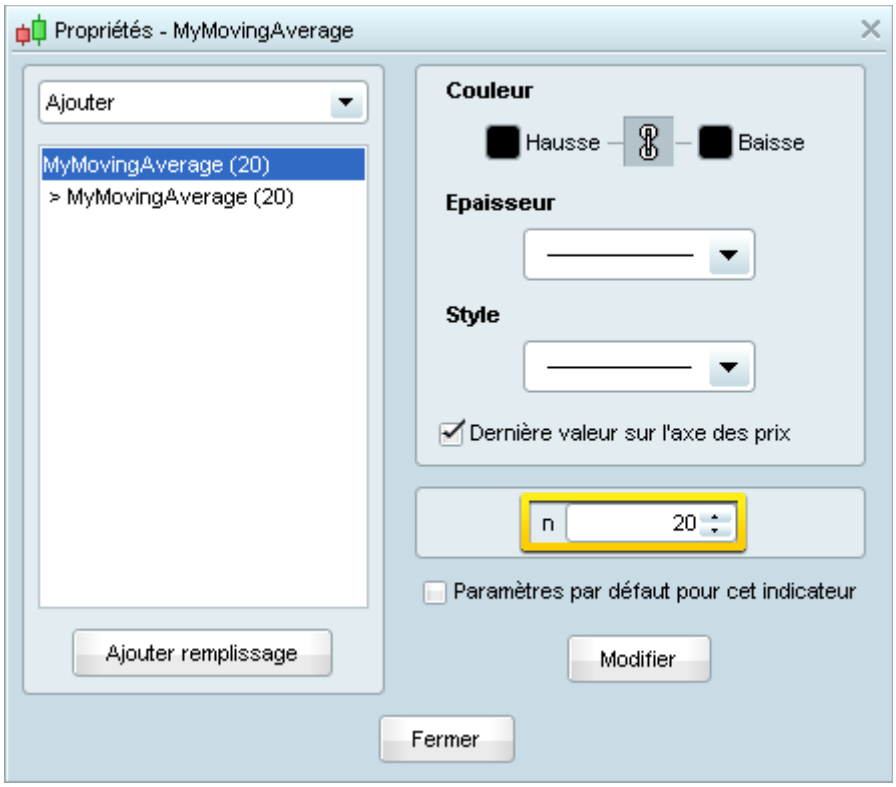
Afin de pouvoir modifier le nombre de périodes de calcul directement à partir de l'interface, remplacez 20 par une variable 'n' :

```
RETURN Average[n](Close)
```

Cliquez ensuite sur "Ajouter" dans "Optimisation des variables" et vous verrez alors s'afficher la fenêtre "Définition de la variable". Remplissez comme suit :



Appuyez sur "OK". Dans la fenêtre de Propriété de l'indicateur (dans ce cas "Propriétés MyMovingAverage"), vous obtiendriez donc un nouveau paramètre, qui vous permettra d'agir sur les nombres de périodes de la moyenne mobile :



Bien entendu, il est possible de faire la même chose avec plusieurs variables vous donnant ainsi la possibilité de "jouer" sur plusieurs paramètres en même temps.

Chapitre II : Fonctions et instructions ProBuilder

Structures de contrôle

Instruction conditionnelle IF

L'instruction **IF** sert à faire un choix d'actions conditionnées, c'est-à-dire à subordonner un résultat à la vérification d'une ou plusieurs conditions définies.

La structure se compose des éléments **IF**, **THEN**, **ELSE**, **ELSIF**, **ENDIF**, qui se combinent selon la complexité des conditions que nous voulons définir. Nous allons en passer en revue le mode d'emploi.

Une condition, un résultat (IF THEN ENDIF)

Nous avons la possibilité de rechercher une condition et de définir une action si la condition est vérifiée. En revanche, si la condition n'est pas satisfaite, rien ne se passe (retourne par défaut 0).

Dans l'exemple, si le dernier prix est supérieur à la MM de période 20, alors on affiche la valeur 1.

```
IF Close > Average[20](Close) THEN
   Result = 1
ENDIF
RETURN Result
```

Si le prix est > à la moyenne mobile de 20 périodes
ALORS Result sera égal à 1
SINON
Result sera égal à 0 (par défaut) FIN DE CONDITION

RETURN doit toujours etre suivi de la variable de stockage utilisée (dans l'exemple, Result) si on veut afficher le résultat de la condition

Une condition, deux résultats (IF THEN ELSE ENDIF)

Nous pouvons également choisir de définir un résultat au cas où la condition n'est pas vérifiée. Reprenons l'exemple précédent : si le dernier prix est supérieur à la MM de période 20, on affiche la valeur 1.

Autrement, on affiche -1.

```
IF Close > Average[20](Close) THEN
   Result = 1
ELSE
   Result = -1
ENDIF
RETURN Result
```

NB : Nous venons de créer un indicateur binaire. Pour en savoir plus, voir la section sur les indicateurs binaires et ternaires plus loin dans ce manuel.

Conditions imbriquées

Il est possible de créer des sous-conditions à la suite de la validation d'une condition principale, c'est-à-dire des conditions qui doivent se vérifier l'une après l'autre (dans l'ordre de parution). Pour ce faire, il suffit d'imbriquer les IF, mais en faisant attention à insérer autant de ENDIF que de IF. Regardons l'exemple :

Double conditions sur moyennes mobiles :

```
IF (Average[12](Close) - Average[20](Close) > 0) THEN
   IF ExponentialAverage[12](Close) - ExponentialAverage[20](Close) > 0 THEN
      Result = 1
   ELSE
      Result = -1
   ENDIF
ENDIF
RETURN Result
```

Conditions Multiples (IF THEN ELSIF ELSE ENDIF)

Il est possible de définir plusieurs résultats associés chacun à une condition spécifique. L'indicateur reporte donc plusieurs états : si la Condition1 est vérifiée alors on active l'Action 1 ; autrement, si la Condition 2 est vérifiée, on active l'Action 2 …si aucune condition n'est vérifiée, on active l'Action n.

Syntaxiquement, cette structure utilise les instructions : **IF**, **THEN**, **ELSIF**, **THEN .... ELSE**, **ENDIF**.

Elle s'écrit de la manière suivante :

```
IF (Condition1) THEN
   (Action1)
ELSIF (Condition2) THEN
   (Action2)
ELSIF (Condition3) THEN
   (Action3)
...
...
...
ELSE
   (Action n)
ENDIF
```

Il est possible, mais l'écriture est plus lourde, de remplacer les **ELSIF** par des **ELSE IF.** Il faudra alors terminer la boucle par autant d'**ENDIF** que d'**IF** écrits. Il vous est donc conseillé, si vous souhaitez imbriquer de multiples conditions dans votre programme, d'utiliser **ELSIF** plutôt que **ELSE IF**.

Exemple : détection des avalements haussiers et baissiers

Cette indicateur va retourner 1 si un avalement haussier est détecté, -1 si un avalement baissier est détecté et 0 le reste du temps.

```
// Description d'un avalement haussier
Condition1 = Close[1] < Open[1]
Condition2 = Open < Close[1]
Condition3 = Close > Open[1]
Condition4 = Open < Close
```

```
// Description d'un avalement baissier
Condition5 = Close[1] > Open[1]
Condition6 = Close < Open
Condition7 = Open > Close[1]
Condition8 = Close < Open[1]
```

```
IF Condition1 AND Condition2 AND Condition3 AND Condition4 THEN
   a = 1
ELSIF Condition5 AND Condition6 AND Condition7 AND Condition8 THEN
   a = -1
ELSE
   a = 0
ENDIF
RETURN a
```

Exemple : pivot Demarks Résistance

```
IF DClose(1) > DOpen(1) THEN
   Phigh = DHigh(1) + (DClose(1) - DLow(1)) / 2
   Plow = (DClose(1) + DLow(1)) / 2
ELSIF DClose(1) < DOpen(1) THEN
   Phigh = (DHigh(1) + DClose(1)) / 2
   Plow = DLow(1) - (DHigh(1) - DClose(1)) / 2
ELSE
   Phigh = DClose(1) + (DHigh(1) - DLow(1)) / 2
   Plow = DClose(1) - (DHigh(1) - DLow(1)) / 2
ENDIF
RETURN Phigh , Plow
```

Exemple : BarIndex

Dans le chapitre I de ce manuel, BarIndex vous a été présenté comme compteur du nombre des barres depuis le début de l'historique affiché. BarIndex est souvent utilisé en association avec IF. Par exemple, si on cherche à savoir si notre graphique contient moins ou plus de 23 barres, on écrira :

```
IF BarIndex <= 23 THEN
   a = 0
ELSIF BarIndex > 23 THEN
   a = 1
ENDIF
RETURN a
```

Boucle itérative FOR

La boucle **FOR** est utilisée lorsqu'on souhaite appeler un par un, une série finie d'éléments. Ces éléments pourront être de toute sorte du moment que la série soit ordonnée.

La structure se compose des **FOR**, **TO, DOWNTO**, **DO**, **NEXT**. L'utilisation de **TO** ou **DOWNTO** varie en fonction de l'appel en ordre croissant ou décroissant des éléments. Il est important de souligner que ce qui se trouve entre le **FOR** et le **DO** sont les bornes de l'intervalle à balayer.

Avancement croissant (FOR, TO, DO, NEXT)

```
FOR (Variable = ValeurDeDebutDeSerie) TO ValeurDeFinDeSerie DO
   (Action)
NEXT
```

Exemple : lissage de Moyenne Mobile de période 12 (MM12)

Nous allons créer une variable de stockage (Result) qui va sommer une par une chaque moyenne mobile, de période 11, 12 et 13.

```
Result = 0
FOR Variable = 11 TO 13 DO
   Result = Average[Variable](Close) + Result
NEXT
REM Faisons la moyenne des moyennes mobiles en divisant Result par 3 et en stockant le résultat dans AverageResult.
AverageResult = Result / 3
RETURN AverageResult
```

Visualisons ce qui se passe étape par étape :

Mathématiquement, on veut faire la moyenne des moyennes mobiles arithmétiques de période 11, 12 et 13.

Variable va donc prendre successivement les valeurs 11, 12 puis 13

Result = 0

Variable = 11

Result reçoit la valeur du précédent Result + MM11 = (0) + MM11 = (0 + MM11)

On passe à la valeur suivante du compteur

Variable = 12

Result reçoit la valeur du précédent Result + MM12 = (0 + M11) + MM12 = (0 + MM11 + MM12)

On passe à la valeur suivante du compteur

Variable = 13

Result reçoit la valeur du précédent Result + MM13 = (0 + M11 + M12) + M13 = (0 + M11 + M12 + M13)

La valeur 13 est la dernière valeur du compteur.

On ferme la boucle "**FOR**" avec l'instruction "**NEXT**".

On affiche Result

Ce code signifie tout simplement que variable va dans un premier temps prendre la valeur de début de série, puis variable prendra la valeur suivante (la précédente + 1) et ainsi de suite jusqu'à ce que variable prenne la valeur de fin de la série. Alors, la boucle se ferme.

Exemple : Moyenne sur les 20 dernières barres des plus hauts

```
IF BarIndex < 20 THEN                 S'il n'y a pas plus de 20 périodes sur l'historique
   MMhigh = Undefined                 Alors on attribue à MMhigh la valeur par défaut "rien"
ELSE                                  Sinon
   FOR i = 0 TO 20 DO                 Pour les valeurs entre 1 et 20
      SUMhigh = High[i]+SUMhigh       On somme les 20 dernières valeurs des plus hauts
   NEXT
ENDIF
MMhigh = SUMhigh / 20                 On moyenne cette somme par 20 et on l'affecte à MMhigh
RETURN MMhigh                         On affiche MMhigh
```

Avancement décroissant (FOR, DOWNTO, DO, NEXT)

L'avancement décroissant utilise en revanche les instructions : **FOR**, **DOWNTO**, **DO**, **NEXT**.

Elle s'écrit de la manière suivante :

```
FOR (Variable = ValeurDeFinDeSérie) DOWNTO ValeurDeDébutDeSérie DO
   (Action)
NEXT
```

Reprenons l'exemple de la moyenne mobile sur les 20 dernières barres des prix les plus hauts :

On remarquera que l'on a juste inversé les bornes de l'intervalle balayé.

```
IF BarIndex = 0 THEN
   MMhigh = Undefined
ELSE
   FOR i = 20 DOWNTO 1 DO
      SUMhigh = High[i] + SUMhigh
   NEXT
ENDIF
MMhigh = SUMhigh / 20
RETURN Mmhigh
MMhigh = SUMhigh / 20
RETURN Mmhigh
```

Boucle conditionnelle WHILE

La boucle **WHILE** sert à appliquer une action tant qu'une condition reste valide. Vous verrez que cette boucle a de grandes similitudes avec l'instruction conditionnelle simple **IF**/**THEN**/**ENDIF**.

Syntaxiquement, cette structure utilise les instructions : **WHILE**,(**DO** facultatif), **WEND**

La structure s'écrit de la manière suivante :

```
WHILE (Condition) DO
    (Action 1)
    …
    (Action n)
WEND
```

Voici ci-après un exemple intuitif :

```
Result = 0
WHILE Close > Average[20](Close) DO
   Result = 1
WEND
RETURN Result
```

Exemple : indicateur calculant le nombre de périodes de hausse consécutives

```
Increase = (Close > Close[1])
Count = 0
WHILE Increase[Count] DO
   Count = Count + 1
WEND
RETURN Count
```

*Remarque générale pour l'instruction conditionnelle **WHILE** :*

*De la même façon que pour **IF**, le programme ne traitera pas la boucle conditionnelle écrite lorsque la condition de validation est inconnue.*

Prenons un exemple :

```
Count = 0
WHILE i <> 11 DO
   i = i + 1
   Count = Count + 1
WEND
RETURN Count
```

L'instruction **WHILE** ne connaît pas la valeur d'origine de i donc ne peut pas tester si i est bien égal à 10.

La boucle va utiliser ses ressources pour définir la variable i et lui donner la valeur 0 par défaut. Count ne sera plus traité d'où la valeur de retour 0.

Le code correct serait :

```
i = 0
Count = 0
WHILE i <> 11 DO
   i = i + 1
   Count = Count + 1
WEND
RETURN Count
```

Dans ce code, i est bien initialisé. La boucle va donc fonctionner correctement car la condition est maintenant valide.

BREAK

L'instruction **BREAK** permet de faire une sortie forcée d'une boucle **WHILE** ou d'une boucle **FOR**. Des combinaisons avec la commande **IF** sont possibles, que ce soit dans une boucle **WHILE** ou dans une boucle **FOR**.

Avec WHILE

Lorsqu'on cherche à sortir d'une boucle conditionnelle **WHILE**, à savoir que l'on n'attend pas de trouver une situation ne satisfaisant pas à la condition de bouclage, on utilise **BREAK** suivant la structure suivante :

```
WHILE (Condition) DO
   (Action)
   BREAK
WEND
```

Prenons pour exemple un indicateur qui cumule le nombre de périodes de hausse et de baisse consécutifs :

```
REM Indicateur de tendance : cumule le nombre de hausses et de baisses
REM On augmente de 1 un compteur initialisé à zéro en cas de hausse (ou -1 pour une baisse)
Increase = (Close - Close[1]) > 0
Indicator = 0
i = 0
WHILE Increase[i] DO
   Indicator = Indicator + 1
   i = i + 1
   BREAK
WEND
RETURN Indicator
```

Dans ce code, si on n'avait pas utilisé **BREAK**, la boucle aurait continué et le résultat serait un autre indicateur de tendance cumulant le nombre de hausses consécutives.

Avec FOR

Lorsqu'on cherche à sortir d'une boucle itérative **FOR**, sans arriver à la dernière (ou première) valeur de la série, on utilise **BREAK** suivant la structure suivante :

```
FOR (Variable = ValeurDeFinDeSerie) TO ValeurDeDebutDeSerie DO
   (Action)
   BREAK
NEXT
```

Prenons pour exemple un indicateur cumulant le nombre de hausses consécutives du volume dans les 19 dernières barres. Cet indicateur rendra zéro si le volume est baissier.

```
Indicator = 0
FOR i = 0 TO 19 DO
   IF (Volume[i] > Volume[i + 1]) THEN
      Indicator = Indicator + 1
   ELSE
      BREAK
   ENDIF
NEXT
RETURN Indicator
```

Dans ce code, si on n'avait pas utilisé **BREAK**, la boucle aurait continué jusqu'à 19 (dernier élément de la série) même si la condition de volume n'est pas valide.

Avec **BREAK**, en revanche, dès que la condition n'est plus validée, il retourne le résultat et revient à 0.

CONTINUE

L'instruction **CONTINUE** permet de replacer la lecture du programme à la ligne de début d'une boucle **WHILE** ou **FOR**. Il est souvent utilisé en association avec **BREAK**, pour donner l'ordre soit de sortir de la boucle (**BREAK**) soit d'y rester (**CONTINUE**).

Avec WHILE

Créons un programme cumulant le nombre de chandeliers ayant une clôture supérieure et une ouverture inférieure à celles de la veille. Si la condition n'est pas vérifiée, le compteur reviendra à zéro.

```
Increase = Close > Close[1]
Count = 0
WHILE Open < Open[1] DO
   IF Increase[Count] THEN
      Count = Count + 1
      CONTINUE
   ENDIF
BREAK
WEND
RETURN Count
```

Grâce à **CONTINUE**, lorsque la condition du **IF** est vérifiée, on ne sort pas de la boucle **WHILE**, ce qui permet de cumuler le nombre de figures vérifiant cette condition. Sans l'instruction **CONTINUE**, le programme sortirait de la boucle, que la condition du **IF** soit vérifiée ou non. On ne pourrait donc pas cumuler les apparitions de figures et le résultat serait binaire (1,0).

Avec FOR

Créons un programme cumulant le nombre de chandeliers ayant une clôture supérieure à la veille. Si la condition n'est pas vérifiée, le compteur reviendra à zéro.

```
Increase = Close > Close[1]
Count = 0
FOR i = 1 TO BarIndex DO
   IF Increase[Count] THEN
      Count = Count + 1
      CONTINUE
   ENDIF
BREAK
NEXT
RETURN Count
```

FOR permet de tester la condition sur tout l'historique disponible. Grâce à **CONTINUE**, lorsque la condition du **IF** est vérifiée, on ne sort pas de la boucle **FOR** et on continue avec la valeur du i suivant. Ceci permet de cumuler le nombre de figures vérifiant cette condition.

Sans l'instruction **CONTINUE**, le programme sortirait de la boucle, que la condition du **IF** soit vérifiée ou non.

On ne pourrait donc pas cumuler les apparitions de figures et le résultat serait binaire (1,0).

ONCE

L'instruction **ONCE** sert à ne déclarer "**qu'une seule fois**" une variable.

Sachant que pour tout programme, le langage va lire autant de fois le code qu'il y a de barres sur le graphique avant de retourner un résultat, il faudra donc retenir que **ONCE** :

- N'est traité par le programme qu'une seule et unique fois, relecture incluse.
- Lors de la relecture du code par le langage, celle-ci va conserver les valeurs calculées à l'issue de la lecture précédente.

Pour bien comprendre comment fonctionne cette commande, il faut percevoir la manière dont le langage lit le code; d'où l'utilité de l'exemple suivant.

Voici deux programmes qui retournent respectivement 0 et 15 et dont la seule différence est l'ajout de la commande **ONCE** :

Programme 1

```
1 Count = 0
2 i = 0
3 IF i <= 5 THEN
4    Count = Count + i
5    i = i + 1
6 ENDIF
7 RETURN Count
```

Programme 2

```
1 ONCE Count = 0
2 ONCE i = 0
3 IF i <= 5 THEN
4    Count = Count + i
5    i = i + 1
6 ENDIF
7 RETURN Count
```

Voyons voir comment le langage a lu les codes.

Programme 1 :

Le langage va lire L1 (Count = 0 ; i = 0), puis L2, L3, L4, L5 et L6 (Count = 0 ; i = 1), revient à L1 et relira tout exactement de la même façon. Avec **RETURN**, le langage opère une sortie du programme après avoir lu "n barres fois" ce dernier. Le résultat affiché est 0 (zéro), comme à la suite de la première lecture.

Programme 2 :

Le langage va lire L1 (Count = 0 ; i = 0), puis L2, L3, L4, L5, L6 (Count = 0 ; i = 1) ; arrivé à la ligne de **RETURN**, il recommence la boucle depuis L3 (**les lignes avec ONCE ne sont traitées que la première fois**), L4, L5, L6 (Count = 1 ; i = 2), puis revient à nouveau (Count = 3 ; i = 3) et ainsi de suite jusqu'à (Count = 15 ; i = 6). Arrivé à ce résultat, la boucle en **IF** n'est plus traitée car la condition ne vaut plus; ne lui restera à lire que L7. D'où le résultat : 15.

Fonctions Mathématiques

Fonctions usuelles unaires et binaires

Intéressons-nous maintenant aux fonctions mathématiques.

A noter que a et b sont des exemples d'arguments décimaux. Ils peuvent être remplacés par n'importe quelle variable dans votre programme.

- **MIN**(a, b) : calcule le minimum de a et de b
- **MAX**(a, b) : calcule le maximum de a et de b
- **ROUND**(a) : calcule un arrondi à l'unité de a
- **ABS**(a) : calcule la valeur absolue de a
- **SGN**(a) : donne le signe de a (1 pour positif, -1 pour négatif)
- **SQUARE**(a) : calcule le carré de a
- **SQRT**(a) : calcule la racine carré de a
- **LOG**(a) : calcule le logarithme népérien de a
- **EXP**(a) : calcule l'exponentiel de a
- **COS**(a) : calcule le cosinus de a
- **SIN**(a) : calcule le sinus de a
- **TAN**(a) : calcule la tangente de a
- **ATAN**(a) : calcule l'arc-tangente de a

Codons par exemple la loi mathématique normale, intéressante car elle utilise à la fois la mise au carré, la mise en racine carrée et l'exponentielle :

```
REM Loi Normale appliquée en x = 10, Ecart-type = 6 et Espérance = 8
REM Posons en variable optimisée :
Ecarttype = 6
Esperance = 8
x = 10
Indicator= EXP((1/2)*(SQUARE(x-Esperance)/Ecarttype))/(Ecarttype*SQRT(2/3.14))
RETURN Indicator
```

Opérateurs mathématiques usuels

- **a < b :** a est strictement inférieur à b
- **a <= b ou a =< b :** a est inférieur ou égal à b
- **a > b :** a est strictement supérieur à b
- **a >= b ou a => b :** a est supérieur ou égal à b
- **a = b :** a est égal à b (ou a reçoit la valeur b)
- **a <> b :** a est différent de b

Fonctions de comparaisons graphiques

- **a CROSSES OVER b :** a franchit b à la hausse
- **a CROSSES UNDER b :** a franchit b à la baisse

Fonctions de sommation

- **cumsum :** Calcule la somme de toutes les barres du graphique

La syntaxe d'utilisation de cumsum est :

```
cumsum (prix ou indicateur)
```

- **summation :** Calcule la somme sur un nombre de barres à définir

La somme est effectuée à partir de la barre la plus récente (de droite à gauche)

La syntaxe d'utilisation de **summation** est :

```
summation[nombre de barres](prix ou indicateur)
```

Fonctions Statistiques

La syntaxe d'utilisation de ces fonctions est la même que celle des indicateurs et de la fonction Sommation à savoir :

```
lowest[nombre de barres](prix ou indicateur)
```

- **lowest :** donne la valeur la plus basse sur la période définie
- **highest :** donne la valeur la plus élevée sur la période définie
- **STD :** donne l'écart-type sur une valeur pour une période définie
- **STE :** donne l'écart-erreur sur une valeur pour une période définie

Opérateurs logiques

De même, comme tout langage informatique, il est nécessaire d'avoir à disposition des opérateurs logiques afin de créer des indicateurs pertinents. Vous trouverez ci-dessous les 4 opérateurs logiques de ProBuilder :

- **NOT**(a) : NON logique
- a **OR** b : OU logique
- a **AND** b : ET logique
- a **XOR** b : OU exclusif

Calcul de l'indicateur de tendance : On Balance Volume (OBV) :

```
IF NOT((Close > Close[1]) OR (Close = Close[1])) THEN
   MyOBV = MyOBV - Volume
ELSE
   MyOBV = MyOBV + Volume
ENDIF
RETURN MyOBV
```

Instructions ProBuilder

- **RETURN :** affiche le résultat
- **CustomClose :** renvoie une valeur de prix paramétrable ; par défaut, elle renvoie "Close"
- **CALL :** appelle une fonction précédemment créée par l'utilisateur
- **AS :** nomme les différents résultats affichés
- **COLOURED :** colorie le tracé affiché d'une couleur à définir

RETURN

Nous avons déjà pu voir dans le premier chapitre, l'importance de l'instruction **RETURN**. Elle a des propriétés particulières qu'il faut connaître pour éviter certaines erreurs de programmation.

Pour une utilisation correcte dans l'écriture d'un programme, **RETURN** s'utilise :

- Une seule et unique fois
- A la dernière ligne de code
- Optionnellement avec d'autres fonctions telles qu'AS et COLOURED
- Pour afficher plusieurs résultats, on écrit RETURN suivi par les résultats qu'on veut afficher séparés par une virgule (exemple : RETURN a,b)

REM ou //

REM permet de placer dans le code des remarques. Elles servent principalement à vous souvenir comment fonctionne une fonction que vous auriez codée. Ces remarques seront lues mais évidemment pas traitées par le code. Illustrons l'idée par l'exemple suivant :

```
REM ce programme retourne la moyenne mobile arithmétique de période 20 sur le prix de clôture
RETURN Average[20](Close)
```

N'utilisez pas les caractères spéciaux (exemples : é,ù,ç,ê…) dans ProBuilder, même dans la section **REM**

CustomClose

CustomClose est une constante qui permet d'afficher les constantes **Close**, **Open**, **High**, **Low** et d'autres valeurs, qui peuvent être sélectionnées dans la fenêtre des propriétés de l'indicateur.

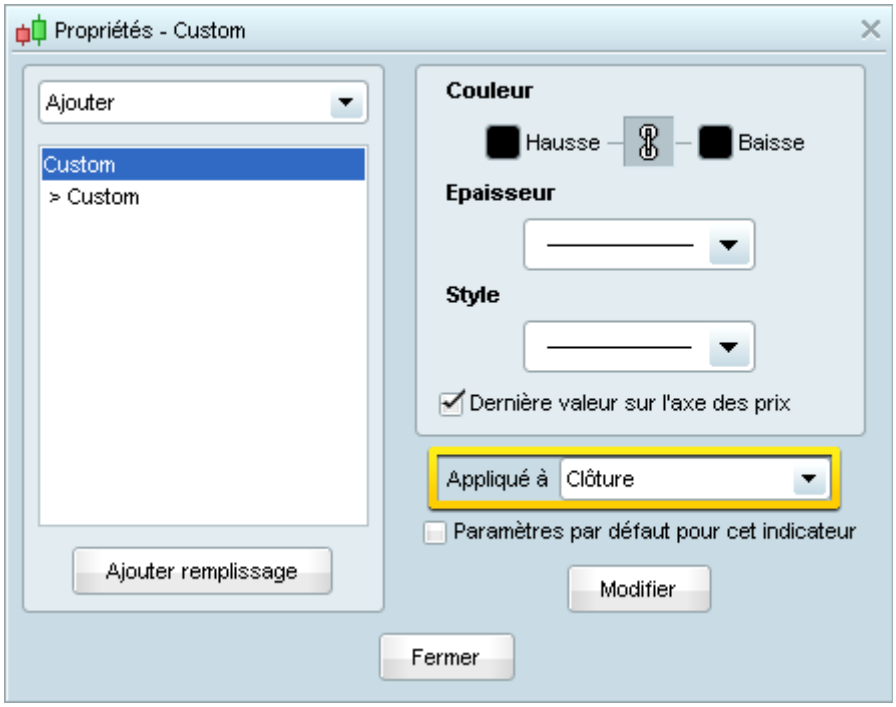
Sa syntaxe d'utilisation est la même que les constantes de prix qui s'adaptent à la vue du graphique :

```
CustomClose[n]
```

Prenons un exemple simple :

```
RETURN CustomClose[2]
```

En cliquant sur la clé à molette sur le coin supérieur gauche du graphique vous verrez qu'il est possible de configurer les prix utilisés pour le calcul (sur le schéma, entouré en jaune).



CALL

CALL permet d'appeler un indicateur personnalisé déjà présent sur votre plateforme.

Le moyen le plus rapide consiste à sélectionner directement à partir de la catégorie "Indicateurs utilisateur" (dans le menu "Insérer fonction") l'indicateur à employer.

Imaginons que vous avez codé sous le nom de HistoMACD l'indicateur du MACD en histogramme.

Sélectionnez votre indicateur et cliquez sur "Ajouter" et dans la zone de programmation apparaîtra :

```
myHistoMACD = CALL HistoMACD
```

Le logiciel a lui-même nommé votre ancien indicateur "HistoMACD" en "myHistoMACD".

Ce qui signifie que pour le reste de votre programme, si vous souhaitez utiliser cet indicateur HistoMACD, vous devrez l'appeler par "myHistoMACD".

AS

Le mot-clé **AS** sert à nommer le résultat affiché. Cette instruction est utilisée avec **RETURN** selon la structure suivante :

```
RETURN Result1 AS "Curve Name", Result2 AS "Curve Name", …
```

L'avantage consiste à faciliter l'identification des éléments composants l'indicateur crée.

Exemple :

```
a = ExponentialAverage[200](Close)
b = WeightedAverage[200](Close)
c = Average[200](Close)
RETURN a AS "Exponential Average", b AS "Weighted Average", c AS "Arithmetical Average"
```

COLOURED

COLOURED est utilisé après la commande **RETURN** pour colorier le tracé affiché d'une certaine couleur, définie selon la norme RGB (red, green, blue). On peut donner les principales couleurs de cette norme :

COULEUR	VALEUR RGB (ROUGE, VERT, BLEU)	FRANÇAIS
	(0, 0, 0)	noir
	(255, 255, 255)	blanc
	(255, 0, 0)	rouge
	(0, 255, 0)	vert
	(0, 0, 255)	bleu
	(255, 255, 0)	jaune
	(0, 255, 255)	cyan
	(255, 0, 255)	magenta

La syntaxe d'utilisation de la commande Coloured est la suivante :

```
RETURN Indicator COLOURED(Red, Green, Blue)
```

La commande **AS** peut être associée à la commande **COLOURED**(., ., .) en respectant l'ordre suivant :

```
RETURN Indicator COLOURED(Red, Green, Blue) AS "Nom De Ma Courbe"
```

Reprenons l'exemple précédent et insérons **COLOURED** à la ligne du "**RETURN**".

```
a = ExponentialAverage[200](Close)
b = WeightedAverage[200](Close)
c = Average[200](Close)
RETURN a COLOURED(255, 0, 0) AS "Exponential Moving Average", b COLOURED(0, 255, 0) AS "WeightedMoving Average", c COLOURED(0, 0, 255) AS "Simple Moving Average"
```

L'image vous montre la personnalisation des couleurs dans le résultat.

FP - TOTAL 37,905 (-0,77%) 1 heure 03 avr. 2013 15:25
1 mois
1 heure
Prix
03 avr. 2013 15:00
Historique ajusté suite aux dividendes
Ouverture 37,870
Plus Haut 37,935
Plus Bas 37,810
Clôture 37,905
Clôture Veille (-0,77%) -0,295
Var unitaire (0,09%) +0,035
© ProRealTime.com Données en temps réel
Coloured
Exponential Moving Average 38,10658
Weighted Moving Average 38,20115
Simple Moving Average 38,24601
NotColoured
Exponential Average 38,10658
Weighted Average 38,20115
Arithmetical Average 38,24601
05 06 07 08 11 12 13 14 15 18 19 20 21 22 25 26 27
03 avr. 2013 15:00

Chapitre III : Applications pratiques

Créer un indicateur binaire ou ternaire : pourquoi et comment ?

Un indicateur binaire ou ternaire est par définition un indicateur ne pouvant retourner que deux ou trois résultats possibles (d'habitude 0 ,1 ou -1). Son utilité principale dans un contexte boursier est de rendre immédiatement identifiable la vérification de la condition qui constitue l'indicateur.

Utilité d'un indicateur binaire ou ternaire :

- Permettre la détection des principales figures de chandeliers japonais
- Faciliter la lecture du graphique lorsqu'on cherche à repérer plusieurs conditions
- Pouvoir mettre des alertes à 1 condition sur un indicateur qui en incorpore plusieurs ➡ vous aurez donc plus d'alertes à disposition !
- Détecter des conditions complexes aussi sur l'historique
- Faciliter la réalisation d'un backtest

Les indicateurs binaires ou ternaires sont construits à l'aide de la fonction IF. Nous vous conseillions la relecture de la section relative avant de continuer la lecture.

Illustrons la création de ces indicateurs pour détecter des patterns de prix :

Indicateur binaire : détection du marteau

```
// Détection du hammer
Hammer = Close>Open AND High = Close AND (Open-Low) >= 3*(Close-Open)
IF Hammer THEN
   Result = 1
ELSE
   Result = 0
ENDIF
RETURN Result AS "Hammer"
```

Indicateur ternaire : détection de croix dorées et croix mortelles

```
a = ExponentialAverage[10](Close)
b = ExponentialAverage[20](Close)
c = 0
// Détection de croix dorée
IF a CROSSES OVER b THEN
   c = 1
ENDIF
// Détection de croix mortelle
IF a CROSSES UNDER b THEN
   c = -1
ENDIF
RETURN c
```

Note : nous avons affiché les moyennes mobiles exponentielles de période 10 et 20 appliquées au prix de clôture pour bien mettre en évidence les correspondances des résultats de l'indicateur.

Vous pouvez retrouver d'autres indicateurs de détection des figures de prix dans la section "Applications Pratiques" plus loin dans ce manuel.

Créer des indicateurs STOP : suivez vos positions en temps réel

Il est possible de créer des indicateurs qui représentent des STOPS, c'est-à-dire des niveaux de sortie potentiels, définis selon des paramètres personnalisés.

Avec le module de création de stratégie ProBackTest, qui fait l'objet d'un manuel de programmation séparé, il vous est possible de définir les niveaux de sortie d'une stratégie. Cependant, la programmation d'un indicateur qui suit le cours d'une valeur est intéressant car :

- Il permet de visualiser le stop en tant que ligne qui se met à jour en temps réel sur le graphique du prix
- Il n'est pas nécessaire de définir des ordres d'achat et de vente (nécessaire dans le cadre d'un ProBackTest)
- Il est possible d'associer des alertes temps réel, pour être tout de suite alerté de la condition

La programmation des Stops vous permettra d'appliquer les principales commandes vues au cours des chapitres précédents.

Par ailleurs, dans le manuel ProBackTest vous pourrez trouver plusieurs exemples de stop à insérer dans des stratégies d'investissement.

Il existe 4 catégories de stops que nous allons passer en revue :

- **STOP prise de bénéfices statique**
- **STOP loss statique**
- **STOP d'inactivité**
- **STOP suiveur (trailing stop)**

Les codes proposés dans les exemples qui suivent représentent des indications pour la construction des indicateurs de stops. Vous devrez nécessairement les personnaliser en utilisant les instructions apprises dans les chapitres précédents.

STOP prise de bénéfices statique

Un *STOP Prise de bénéfices* ou *Take-Profit*, désigne une borne supérieure de sortie de position. Cette borne est par définition fixe. L'utilisateur de ce STOP prend alors ses bénéfices.

L'indicateur codé ci-dessous indique deux niveaux avec prise de position à la date "Start".

- si vous êtes acheteur, vous tiendrez compte de la courbe du haut représentant 10% de gain, soit 110% du cours au moment de l'achat.
- si vous êtes vendeur à découvert, vous tiendrez compte de la courbe du bas représentant aussi 10% de gain, soit 90% du cours au moment de la vente.

En code, cela donne :

Nous vous proposons ci-dessous un exemple de STOP à personnaliser :

```
// On définit en variable optimisée :
// StartingTime = 100000
// Ajustez correctement cette variable à l'heure de votre entrée de position
// Price= Prix au moment de la prise de position (on a pris l'exemple d'une date d'entrée de position définie à 10 heures)
// Si vous êtes en position longue, vous aurez à regarder la courbe du dessus. Si vous êtes en position courte, vous regarderez la courbe du dessous.
// AmplitudeUp représente le taux de variation de Price utilisé pour tracer le Take Profit en position longue (par defaut 1.1)
// AmplitudeDown représente le taux de variation de Price utilisé pour tracer le Take Profit en position courte (par défaut : 0.9)
IF Time = StartingTime THEN
   StopLONG = AmplitudeUp * Price
   StopSHORT = AmplitudeDown * Price
ENDIF
RETURN StopLONG COLOURED(0, 0, 0) AS "TakeProfit LONG", StopSHORT COLOURED(0, 255, 0) AS "TakeProfit SHORT"
```

STOP loss statique

Un *STOP Loss* est le contraire du *STOP Take-Profit* à savoir qu'au lieu de positionner une borne supérieure de sortie, il positionne une borne inférieure. Ce STOP est utile pour limiter les pertes à un seuil minimum.

Comme le *Take-Profit*, ce STOP définit une limite fixe.

L'indicateur codé ci-dessous indique deux niveaux avec prise de position à la date "Start".

- si vous êtes acheteur, vous tiendrez compte de la courbe du bas représentant 10% de perte, soit 90% du cours au moment de l'achat.
- si vous êtes vendeur à découvert, vous tiendrez compte de la courbe du haut représentant aussi 10% de perte, soit 110% du cours au moment de la vente.

En code, cela donne :

```
// On définit en variable optimisée :
// StartingTime = 100000 (on a pris l'exemple d'une date d'entrée de position définie à 10 heures)
// Ajustez correctement cette variable à l'heure de votre entrée de position
// Price= prix d'ouverture de la position
// AmplitudeUp représente le taux de variation de Price utilisé pour tracer le Stop Loss en position longue (par defaut : 0.9)
// AmplitudeDown représente le taux de variation de Price utilisé pour tracer le Stop Loss en position courte (par défaut : 1.1)
IF Time = StartingTime THEN
   StopLONG = AmplitudeUp * Price
   StopSHORT = AmplitudeDown * Price
ENDIF
RETURN StopLONG COLOURED(0, 0, 0) AS "StopLoss LONG", StopSHORT COLOURED(0, 255, 0) AS "StopLoss SHORT"
```

STOP d'inactivité

Un *STOP d'inactivité* ferme la position lorsque les gains n'ont pas atteint un objectif (défini en % ou en points) sur une période définie (exprimée en nombre de barres)

Exemple de STOP d'inactivité en graphique Intraday :

Ce stop est à utiliser avec deux indicateurs :

- le premier est à juxtaposer sur la courbe des prix
- le deuxième à visualiser séparément

Indicateur1

```
// MyVolatility = 0.01 correspond à l'écart relatif des bandes hautes et basse du range défini
IF IntradayBarIndex = 0 THEN
   ShortTarget = (1 - MyVolatility) * Close
   LongTarget = (1 + MyVolatility) * Close
ENDIF
RETURN ShortTarget AS "ShortTarget", LongTarget AS "LongTarget"
```

Indicateur2

```
// On définit en variable optimisée :
REM La prise de position se fait au prix du marché
// MyVolatility = 0.01 correspond à l'écart relatif des bandes hautes et basses du range défini
// NumberOfBars = 20 correspond à la durée maximale en nombre de barres sur lequel peut évoluer les prix avant que les positions ne soient coupées (résultat placé à 1)
Result = 0
Cpt = 0
IF IntradayBarIndex = 0 THEN
   ShortTarget = (1 - MyVolatility) * Close
   LongTarget = (1 + MyVolatility) * Close
ENDIF
FOR i = IntradayBarIndex DOWNTO 1 DO
   IF Close[i] >= ShortTarget AND Close[i] <= LongTarget THEN
      Cpt = Cpt + 1
   ELSE
      Cpt = 0
   ENDIF
   IF Cpt = NumberOfBars THEN
      Result = 1
   ENDIF
NEXT
RETURN Result
```

STOP suiveur ou trailing stop

Un *STOP suiveur* ou *trailing STOP* suit dynamiquement l'évolution des prix et indique à quel moment la position doit être coupée.

Nous vous suggérons ci-dessous deux formes de STOP suiveur, celle correspondant au Stop Loss dynamique, l'autre au Take Profit dynamique.

Trailing STOP LOSS (à utiliser en trading intraday)

```
// On définit en variable optimisée :
// StartingTime = 090000 (on a pris l'exemple d'une date d'entrée de position définie à 9 heures; Ajustez correctement cette variable à l'heure de votre entrée de position)
REM La prise de position se fait au prix du marché
// Amplitude représente le taux de variation des courbes "Cut" avec les courbes "Lowest" (par exemple, on peut prendre Amplitude = 0.95)
IF Time = StartingTime THEN
   IF lowest[5](Close) < 1.2 * Low THEN
      IF lowest[5](Close) >= Close THEN
         Cut = Amplitude * lowest[5](Close)
      ELSE
         Cut = Amplitude * lowest[20](Close)
      ENDIF
   ELSE
      Cut = Amplitude * lowest[20](Close)
   ENDIF
ENDIF
RETURN Cut AS "Trailing Stop Loss"
```

Trailing STOP Profit (à utiliser en trading intraday)

```
// On définit en variable optimisée :
// StartingTime = 090000 (on a pris l'exemple d'une date d'entrée de position définie à 9 heures; Ajustez correctement cette variable à l'heure de votre entré de position)
REM La prise de position se fait au prix du marché
// Amplitude représente le taux de variation des courbes "Cut" avec les courbes "Lowest" (par exemple, on peut prendre Amplitude = 1.015)
IF Time = StartingTime THEN
   StartingPrice = Close
ENDIF
Price = StartingPrice - AverageTrueRange[10]
TrailingStop = Amplitude * highest[15](Price)
RETURN TrailingStop COLOURED (255, 0, 0) AS "Trailing take profit"
```

Chapitre IV : exercices

Figures de chandeliers

- **GAP UP ou DOWN**

La couleur des chandeliers n'a pas d'importance

On définit en variable optimisée l'amplitude égale à 0.001

Un gap se définit par deux conditions :

- (l'ouverture du jour est strictement supérieure à la clôture de la veille) ou (l'ouverture du jour est strictement inférieure à la clôture de la veille)
- la valeur absolue de ((l'ouverture du jour – la clôture de la veille) / la clôture de la veille) est strictement supérieure à l'amplitude

```
// Initialisation de l'Amplitude (du gap)
Amplitude = 0.001
// Initialisation du détecteur
Detector = 0
// Gap Up
// 1ère condition d'existence d'un gap
IF Low > High[1] THEN
   // 2e condition d'existence d'un gap
   IF ABS((Low - High[1]) / High[1]) > Amplitude THEN
      // Comportement du détecteur
      Detector = 1
   ENDIF
ENDIF
// Gap Down
// 1ère condition d'existence d'un gap
IF High < Low[1] THEN
   // 2e condition d'existence d'un gap
   IF ABS((High - Low[1]) / Low[1]) > Amplitude THEN
      // Comportement du détecteur
      Detector = -1
   ENDIF
ENDIF
// Affichage du résultat
RETURN Detector AS "Gap detection"
```

- **Doji (version souple)**

On définit le doji par un range strictement supérieur à 5 fois la valeur absolue de (Open – Close).

```
Doji = Range > ABS(Open - Close) * 5
RETURN Doji AS "Doji"
```

- **Doji (version stricte)**

On définit le doji par un close égal à l'Open.

```
Doji = (Open = Close)
RETURN Doji AS "Doji"
```

Indicateurs

- **BODY MOMENTUM**

Le Body Momentum est défini mathématiquement par :

BodyMomentum = 100 * BodyUp / (BodyUp + BodyDown)

BodyUp (resp. BodyDown) est un compteur du nombre de barres clôturant plus haut (resp. bas) que son ouverture, et ce sur une période définie (prenons période = 14)

```
Periods = 14
b = Close - Open
IF BarIndex > Periods THEN
   Bup = 0
   Bdn = 0
   FOR i = 1 TO Periods
      IF b[i] > 0 THEN
         Bup = Bup + 1
      ELSIF b[i] < 0 THEN
         Bdn = Bdn + 1
      ENDIF
   NEXT
   BM = (Bup / (Bup + Bdn)) * 100
ELSE
   BM = Undefined
ENDIF
RETURN BM AS "Body Momentum"
```

ELLIOT WAVE OSCILLATOR

L'oscillateur Elliot représente la différence entre deux moyennes mobiles.

La moyenne mobile courte représente l'action des prix tandis que la moyenne longue représente la tendance de fond.

Lorsque les prix forment une vague 3, les prix grimpent fortement ce qui produit une valeur importante sur l'oscillateur.

Dans une vague 5, les prix grimpent plus lentement et l'oscillateur donne alors une valeur beaucoup plus faible.

```
RETURN Average[5](MedianPrice) - Average[35](MedianPrice) AS "Elliot Wave Oscillator"
```

Williams %R

Voici un oscillateur dont le fonctionnement est similaire à l'oscillateur stochastique. Pour le tracer, on commence par définir dans un premier temps 2 courbes :

1) la courbe des plus hauts du plus haut sur 14 périodes

2) la courbe des plus bas du plus bas sur 14 périodes

Le %R se définit alors par (Close – LowestL) / (HighestH – LowestL) * 100

```
HighestH = highest[14](High)
LowestL = lowest[14](Low)
MyWilliams = (Close - LowestL) / (HighestH - LowestL) * 100
RETURN MyWilliams AS "Williams %R"
```

Bandes de Bollinger

On définit ces bandes par un encadrement de la moyenne mobile arithmétique sur 20 barres appliquée au prix de clôture.

La moyenne mobile est bornée au-dessus (resp. en-dessous) par + (resp. -) 2 fois l'écart-type prise sur les 20 barres précédentes du prix de clôture.

```
a = Average[20](Close)
// On définit l'écart-type
StdDeviation = STD[20](Close)
Bsup = a + 2 * StdDeviation
Binf = a - 2 * StdDeviation
RETURN a AS "Average", Bsup AS "Bollinger Up", Binf AS "Bollinger Down"
```

Glossaire

A

CODE	IMPLÉMENTATION	FONCTION
ABS	ABS(a)	Fonction Mathématique "Valeur Absolue"
AccumDistr	AccumDistr(price)	Désigne l'Accumulation Distribution classique
ADX	ADX[N]	Indicateur Average Directional Index
ADXR	ADXR[N]	Indicateur Average Directional Index Rate
AND	a AND b	Opérateur logique ET
AroonDown	AroonDown[P]	Désigne l'Aroon Down
AroonUp	AroonUp[P]	Désigne l'Aroon Up
ATAN	ATAN(a)	Fonction mathématique "Arc tangente"
AS	RETURN x AS "ResultName"	Instruction servant à nommer une courbe
Average	Average[N](price)	Moyenne Mobile Arithmétique
AverageTrueRange	AverageTrueRange[N](price)	Désigne la moyenne mobile par lissage de Wilder du True Range

B

CODE	IMPLÉMENTATION	FONCTION
BarIndex	BarIndex	Nombre de barres depuis la première barre de données chargée (dans un graphique dans le cas d'un indicateur ProBuilder ou pour un système de trading dans le cas d'un ProBacktest ou ProInvest)
BollingerBandWidth	BollingerBandWidth[N](price)	Bande passante de Bollinger
BollingerDown	BollingerDown[N](price)	Support de la bande de Bollinger
BollingerUp	BollingerUp[N](price)	Résistance de la bande de Bollinger
BREAK	(FOR...DO...BREAK...NEXT) ou (WHILE...DO...BREAK...WEND)	Instruction de sortie forcée de boucle FOR ou WHILE

C

CODE	IMPLÉMENTATION	FONCTION
CALL	myResult = CALL myFunction	Appel de fonction utilisateur
CCI	CCI[N](price) ou CCI[N]	Donne le Commodity Channel Index
ChaikinOsc	ChaikinOsc[Ch1, Ch2](price)	Désigne l'oscillateur de Chaikin
Chandle	Chandle[N](price)	Désigne le Chande Momentum Oscillator
ChandeKrollStopUp	ChandeKrollStopUp[Pp, Qq, X]	Stop de protection selon Chande et Kroll en position acheteuse
ChandeKrollStopDown	ChandeKrollStopDown[Pp, Qq, X]	Stop de protection selon Chande et Kroll en position vendeuse
Close	Close[N]	Désigne le prix de clôture de la barre courante ou de celle n jours auparavant
COLOURED	RETURN x COLOURED(R,G,B)	Colorie une courbe d'une certaine couleur selon la convention RGB
COS	COS(a)	Fonction cosinus
CROSSES OVER	a CROSSES OVER b	Opérateur booléen vérifiant qu'une courbe passe au-dessus d'une autre
CROSSES UNDER	a CROSSES UNDER b	Opérateur booléen vérifiant qu'une courbe passe en dessous d'une autre
cumsum	cumsum(price)	Sommation d'un prix depuis le début de l'historique affiché
CurrentDayOfWeek	CurrentDayOfWeek	Désigne le jour actuel
CurrentHour	CurrentHour	Désigne l'heure actuelle
CurrentMinute	CurrentMinute	Désigne la minute actuelle
CurrentMonth	CurrentMonth	Désigne le mois actuel
CurrentSecond	CurrentSecond	Désigne la seconde actuelle
CurrentTime	CurrentTime	Désigne HeureMinute actuelle
CurrentYear	CurrentYear	Désigne l'année actuelle
CustomClose	CustomClose[N]	Constante paramétrable dans la fenêtre de propriétés
Cycle	Cycle(price)	Indicateur Cycle

D

CODE	IMPLÉMENTATION	FONCTION
Date	Date[N]	Désigne la date de clôture de la barre courante
Day	Day[N]	Jour de clôture de la barre courante
Days	Days[N]	Compteur de jours depuis 1900
DayOfWeek	DayOfWeek[N]	Désigne le jour de la semaine durant lequel la barre courante a clos
DClose	DClose(N)	Prix de clôture de la n-ième journée antérieure à celle de la barre courante
DEMA	DEMA[N](price)	Double Moyenne Mobile Exponentielle
DHigh	DHigh(N)	Prix le plus haut de la n-ième journée antérieure à celle de la barre courante
DI	DI[N](price)	Désigne le Demand Index
DIminus	DIminus[N](price)	Désigne le DI-
DIplus	DIplus[N](price)	Désigne le DI+
DLow	DLow(N)	Prix le plus bas de la n-ième journée antérieure à celle de la barre courante
DO	Voir FOR et WHILE	Instruction facultative des FOR et WHILE pour l'action de bouclage
DOpen	DOpen(N)	Prix d'ouverture de la n-ième journée antérieure à celle de la barre courante
DOWNTO	Voir FOR	Instruction sur boucle FOR pour une lecture décroissante
DPO	DPO[N](price)	Désigne le Detrented Price Oscillator

E

CODE	IMPLÉMENTATION	FONCTION
EaseOfMovement	EaseOfMovement[I]	Désigne l'indicateur Ease of Movement
ELSE	Voir IF/THEN/ELSE/ENDIF	Instruction d'appel de la seconde condition à défaut de la première issue du IF
ELSEIF	Voir IF/THEN/ELSE/ENDIF	Contraction de ELSE IF
EMV	EMV[N]	Désigne l'indicateur Ease of Movement Value
ENDIF	Voir IF/THEN/ELSE/ENDIF	Instruction de clôture des instructions conditionnelles
EndPointAverage	EndPointAverage[N](price)	Moyenne Mobile à dernier point
EXP	EXP(a)	Fonction Mathématique "Exponentielle"
ExponentialAverage	ExponentialAverage[N](price)	Moyenne Mobile Exponentielle

F - G

CODE	IMPLÉMENTATION	FONCTION
FOR/TO/NEXT	FOR i =a TO b DO a NEXT	Prend les valeurs désignées du début à la fin ou vice versa
ForceIndex	ForceIndex(price)	Indicateur Force Index déterminant qui contrôle le marché

H

CODE	IMPLÉMENTATION	FONCTION
High	High[N]	Désigne le plus haut cours atteint durant la période N
highest	highest[N](price)	Désigne le plus haut cours sur un horizon donné
HistoricVolatility	HistoricVolatility[N](price)	Désigne la volatilité historique ou statistique
Hour	Hour[N]	Désigne l'heure de clôture de chaque barre

I - J - K

CODE	IMPLÉMENTATION	FONCTION
IF/THEN/ENDIF	IF a THEN b ENDIF	Ensemble d'instructions conditionnelles sans deuxième condition
IF/THEN/ELSE/ENDIF	IF a THEN b ELSE c ENDIF	Ensemble d'instructions conditionnelles
IntradayBarIndex	IntradayBarIndex[N]	Compte le nombre de chandeliers sur le graphique intraday

L

CODE	IMPLÉMENTATION	FONCTION
LinearRegression	LinearRegression[N](price)	Droite de régression linéaire
LinearRegressionSlope	LinearRegressionSlope[N](price)	Pente de la droite de régression linéaire
LOG	LOG(a)	Fonction mathématique "logarithme népérien"
Low	Low[N]	Désigne le plus bas atteint durant la période
lowest	lowest[N](price)	Désigne le plus bas d'une période sur un horizon donné

M

CODE	IMPLÉMENTATION	FONCTION
MACD	MACD[S,L,Si](price)	Moving Average Convergence Divergence (MACD)
MACDline	MACDLine[S,L](price)	Désigne la ligne du MACD
MassIndex	MassIndex[N]	Indicateur Mass Index appliqué sur N barres
MAX	MAX(a,b)	Fonction mathématique "Maximum"
MedianPrice	MedianPrice	Moyenne du prix le plus haut et du plus bas
MIN	MIN(a,b)	Fonction Mathématique "Minimum"
Minute	Minute	Désigne la minute du moment de la clôture de chaque barre de l'historique
MOD	a MOD b	Fonction Mathématique "Reste de la division euclidienne"
Momentum	Momentum[l]	Momentum (prix de clôture – prix de clôture de la n-ième barre précédente)
MoneyFlow	MoneyFlow[N](price)	Donne le MoneyFlow entre -1 et 1
MoneyFlowIndex	MoneyFlowIndex[N]	Désigne le MoneyFlowIndex
Month	Month[N]	Désigne le mois de la clôture de chaque barre de l'historique

N

CODE	IMPLÉMENTATION	FONCTION
NEXT	Voir FOR/TO/NEXT	Instruction à placer à la fin de la boucle "FOR"
NOT	NOT a	Opérateur logique NON

O

CODE	IMPLÉMENTATION	FONCTION
OBV	OBV(price)	Désigne l' "On-Balance-Volume"
ONCE	ONCE VariableName = VariableValue	Instruction qui en précède une autre qu'on ne veut réaliser qu'une seule fois
Open	Open[N]	Désigne le prix d'ouverture de la barre courante ou celle de n jours auparavant
OR	a OR b	Opérateur logique OU

P - Q

CODE	IMPLÉMENTATION	FONCTION
PriceOscillator	PriceOscillator[S,L](price)	Indicateur Percertage Price oscillator
PositiveVolumeIndex	PriceVolumeIndex(price)	Désigne l'indicateur Positive Volume Index
PVT	PVT(price)	Désigne l'indicateur "Price Volume Trend"

R

CODE	IMPLÉMENTATION	FONCTION
R2	R2[N](price)	Coefficient R Carré (taux d'erreur des prix à la regression linéaire)
Range	Range[N]	Différence entre le prix le plus haut et le plus bas de la barre courante
REM	REM comment	Précède une remarque dans le code
Repulse	Repulse[N](price)	Mesure la poussée haussière et baissière de chaque bougie
RETURN	RETURN Result	Instruction qui renvoie le résultat
ROC	ROC[N](price)	Désigne le "Price Rate of Change"
RSI	RSI[N](price)	Désigne l'oscillateur "Relative Strength Index"
ROUND	ROUND(a)	Fonction mathématique "Arrondi à l'unité"

S

CODE	IMPLÉMENTATION	FONCTION
SAR	SAR[At,St,Lim]	Désigne le Parabolique SAR
SARatdmf	SARatdmf[At,St,Lim](price)	Parreil (parabolique SAR) Désigne le Parabolique SAR ATDMF
SIN	SIN(a)	Fonction Mathématique "Sinus"
SGN	SGN(a)	Fonction Mathématique "Signe de"
SMI	SMI[N,SS,DS](price)	Désigne le Stochastic Momentum Index
SmoothedStochastic	SmoothedStochastic[N,K](price)	Désigne une Stochastique lissée
SQUARE	SQUARE(a)	Fonction mathématique "Mise au carré"
SQRT	SQRT(a)	Fonction Mathématique "Mise à la racine carrée"
STD	STD[N](price)	Fonction Statistique "écart-type"
STE	STE[N](price)	Fonction Statistique "écart-erreur"

Stochastic	Stochastic[N,K](price)	Ligne %K de la Stochastique
summation	summation[N](price)	Somme d'un certain prix des N derniers chandeliers
Supertrend	Supertrend[STF,N]	Désigne le Super Trend

T

CODE	IMPLÉMENTATION	FONCTION
TAN	TAN(a)	Fonction mathématique "Tangente"
TEMA	TEMA[N](price)	Moyenne Mobile Exponentielle Triple
THEN	Voir IF/THEN/ELSE/ENDIF	Instruction suivant la première condition de l'instruction "IF"
Time	Time[N]	Donne l'HeureMinuteSeconde et permet de faire appel à l'heure
TimeSeriesAverage	TimeSeriesAverage[N](price)	Moyenne mobile des séries temporelles
TO	Voir FOR/TO/NEXT	Instruction "jusqu'à" dans la boucle "Pour"
Today	Today	Date de la journée actuelle
TotalPrice	TotalPrice[N]	(Clôture + Ouverture + Plus Haut + Plus Bas) / 4
TR	TR(price)	Désigne le True Range
TriangularAverage	TriangularAverage[N](price)	Moyenne Mobile Triangulaire
TRIX	TRIX[N](price)	Triple Moyenne Mobile Exponentielle
TypicalPrice	TypicalPrice[N]	Prix Typique (moyenne de plus haut, plus bas et clôture)

U

CODE	IMPLÉMENTATION	FONCTION
Undefined	a = Undefined	Pour laisser une variable indéfinie (Null)

V

CODE	IMPLÉMENTATION	FONCTION
Variation	Variation(price)	Différence entre la clôture de la veille et la clôture courante en %
Volatility	Volatility[S, L]	Désigne la volatilité de Chaikin
Volume	Volume[N]	Désigne le volume
VolumeOscillator	VolumeOscillator[S,L]	Désigne l'oscillateur de volume
VolumeROC	VolumeROC[N]	Désigne le volume du Rate Of Change

W

CODE	IMPLÉMENTATION	FONCTION
WeightedAverage	WeightedAverage[N](price)	Désigne la Moyenne Mobile Pondérée
WeightedClose	WeightedClose[N]	Moyenne pondérée entre le prix de clôture, le plus haut et la plus bas
WEND	Voir WHILE/DO/WEND	Instruction à placer à la fin de la boucle Tant Que
WHILE/DO/WEND	WHILE (condition) DO (action) WEND	Boucle "Tant Que"
WilderAverage	WilderAverage[N](price)	Donne la moyenne mobile de Wilder
Williams	Williams[N](close)	Calcule le %R de Williams
WilliamsAccumDistr	WilliamsAccumDistr(price)	Indicateur Accumulation/Distribution de Williams

X

CODE	IMPLÉMENTATION	FONCTION
XOR	a XOR b	Opérateur logique OU exlusif

Y

CODE	IMPLÉMENTATION	FONCTION
Year	Year[N]	Donne l'évolution des années
Yesterday	Yesterday[N]	Donne l'évolution du jour d'avant

Z

CODE	IMPLÉMENTATION	FONCTION
ZigZag	ZigZag[Zr](price)	Zig-Zag de la théorie des vagues d'Eliott
ZigZagPoint	ZigZagPoint[Zp](price)	Zig-Zag de la théorie des vagues d'Eliott calculé à Zp points

Autres

CODE	FONCTION	CODE	FONCTION
+	Opérateur d'addition	<>	Opérateur de différence
-	Opérateur de soustraction	<	Opérateur d'infériorité strict
*	Opérateur de multiplication	>	Opérateur de supériorité strict
/	Opérateur de division décimale	<=	Opérateur d'infériorité
=	Opérateur d'égalité	>=	Opérateur de supériorité

ProRealTime SOFTWARE